13. Seriële bussen

In dit hoofdstuk gaan we gegevens doorsturen tussen de controller en randapparaten. Soms is dit heel eenvoudig: voor het uitlezen van een schakelaar of het aansturen van een LED volstaan twee draadjes: een draad voor gegevens (aan/uit) en een massadraad. Sommige toepassingen zijn complexer: voor de aansturing van een beeldscherm of het uitlezen van een temperatuursensor zijn gegevens en gegevenstransport uitgebreider: die gegevens zijn meestal gegroepeerd per 8 bits. De overdracht kan op twee manieren: **parallel** of **serieel**. Parallel transport verstuurt de gegevens tegelijk over acht datalijnen en één massadraad, serieel transport verstuurt de acht bits één voor één door dezelfde draad. Het grote voordeel van serieel transport is een eenvoudiger bekabeling. Het printontwerp en de bekabeling in het eindproduct zijn ook een stuk eenvoudiger.

Er bestaan heel wat vormen van seriële dataverbinding. De meeste controllers hebben er enkele van ingebouwd, andere kunnen we softwarematig implementeren. Enkele voorbeelden:

De seriële poort

De oudste vorm van serieel transport is de klassieke **seriële poort** of ***UART poort.*** De RP 2040 heeft 2 seriële poorten. Die worden vooral gebruikt voor de verbinding met de PC. De controller stuurt gegevens voor de PC langs een UART naar een serieel/USB omzetter. Via een USB-kabel gaan ze naar een PC, en worden ze terug naar een serieel signaal omgezet.

De 1-Wire Bus

De 1-Wire Bus is een ontwikkeling van Dallas-Semiconductors (overgenomen door Maxim). Deze bus verbindt één **master** (een controller) met een of meerdere **slaves** (periferiechips). Zoals de naam doet vermoeden gebruikt deze bus één datalijn (én een massalijn) voor de communicatie. Sommige **slaves,** zoals de DS18B20, kunnen hun voeding afleiden uit de datalijn zodat een voedingsdraad kan vervallen. Elke slave heeft een eigen unieke 64-bit code zodat de master kan samenwerken met meerdere slaves. De 1-Wire Bus gebruikt module **onewire**, dat is een standaard onderdeel van MicroPython. De constructor voor een object is

ow = onewire.OneWire(Pin(12))

Dit werkt met elke beschikbare GPIO. Met methode scan() gaat na welke periferie aan de bus hangt. Het programma hieronder scant een 1-wire bus aan GPIO 4.

```
from machine import Pin
import onewire, time, ds18x20
ow = onewire.OneWire(Pin(4))
```

```
print(ow.scan())
```

```
Shell

MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
 [bytearray(b'(\xcb\x85E\x92\x0f\x02\xc2')]
>>>
>>>
```

Figuur 72: De 1-wire bus scannen

Er is één device aangesloten. Heel veel wordt de bus niet meer gebruikt, alleen de DS18B20, een temperatuursensor met 1-wire bus is nog vrij populair. We zien die in hoofdstuk *Sensoren*.

De CAN-bus

De CAN-bus, Controller Area Network, is expliciet ontworpen voor omgevingen met veel elektromagnetische storingen. De eerste toepassing voor CAN was de automobielsector, maar ondertussen is CAN ook doorgedrongen in andere embedded toepassingen, zoals machinesturingen en robots. De CAN-Bus wordt meestal softwarematig geïmplementeerd.

De SPI-bus

De SPI-bus is een seriële verbinding tussen één master en tenminste één slave. De communicatie tussen de master en de slave is full duplex, in twee richtingen tegelijk. De master bestuurt de bus. Voor de communicatie zijn er vier verbindingen nodig: SCLK (de klok), MOSI (master out, slave in), MISO (master in, slave out) en SS of CE (slave select of chip enabled). Er bestaan geen adressen bij SPI, elke slave heeft een eigen SS-verbinding met de master, de andere verbindingen zijn gemeenschappelijk. SS is het signaal dat de bus toewijst aan een slave. De RP2040 heeft 2 SPI-bussen.

Klasse SPI

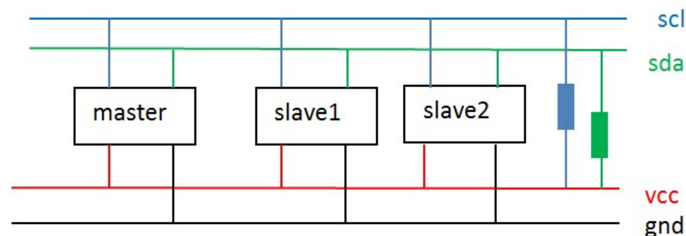
Klasse SPI is een onderdeel van library machine. De constructor is:

spi = machine.SPI(*id*, baudrate= *bbb*, sck = psc, mosi = *pmo*, miso = *pmi*)

id is 0, of 1, het nummer van de SPI, psc, pmo en pmi zijn pinnummers. We zien latere nog voorbeelden met de SPI-bus.

De I2C-bus

De *I2C-bus*, *Inter IC Bus*, is in 1979 ontwikkeld en gepatenteerd door Philips als goedkope verbindingsbus tussen processor en periferie in consumentenelektronica. I2C is gestandaardiseerd in 1990 en andere fabrikanten hebben toen deze technologie geïmplementeerd in hun chips. Ze noemen ze soms T*WI, Two Wire Interface*. De verbinding gebeurd met twee buslijnen: *sda, Serial Data*, verstuurd data en *scl, Serial CLock* het kloksignaal. De bus bestaat uit één **master** en minimaal één **slave**. De master, meestal de controller, stuurt de bus aan. Slaves communiceren alleen over de bus als de master het hen verzoekt. Slaves hebben een 7 bit adres op de bus. Met 7 bits maak je de binaire getallen 0000000 tot 1111111, of decimale getallen 0 tot 127. Adres 0 wordt niet gebruikt, er blijven 127 adressen over voor maximaal 127 slaves. De oorspronkelijke I^2C-bus werkt met een snelheid van 100 kbit per seconde, latere versies gebruiken 400 kb/s of 3.4 Mb/s en hebben een adresseringsruimte van 10 bit.



Figuur 73: I2C systeem, 1 master, 2 of meer slaves

Sla en scl zijn met pull-up weerstanden van ongeveer 1 kΩ verbonden met de voedingslijn. Je kunt ze weglaten bij een systeem met één master en één slave.

De **RP2040** heeft twee I2C poorten: poort I2C -0 en I2C -1. Op het pinout-schema van het bordje zie je met welke GPIO's die verbonden zijn. Voor de Pico zijn twaalf combinaties mogelijk.

	RP2040	
	SDA	**SCL**
I2C0	GP0	GP1
	GP4	GP5
	GP8	GP9
	GP16	GP17
	GP20	GP21
I2C1	GP2	GP3
	GP6	GP7
	GP10	GP11
	GP14	GP15
	GP18	GP19
	GP26	GP27

Waarom twee I2C-bussen? Op een I2C-bus kan je 127 slaves aansluiten, maar elke slave moet een eigen uniek adres hebben. Die adressen zijn vast ingebakken in de slaves. Twee identieke temperatuursensoren aansluiten op één bus kan niet, je moet dan de ene aansluiten op bus I2C0, de andere op bus I2C1 om adresconflicten te vermijden.

De **ESP32** heeft 2 I2C-poorten: I2C(0) en I2C(1). Je mag elk tweetal GPIO's gebruiken voor (sda, scl). De standaard waarden zijn

	I2C(0)	**I2C(1)**
sda	19	26
scl	18	25

Je declareert een I2C poort als

```
from machine import Pin, I2C
i2c = I2C(1)
```

of

```
from machine import Pin, I2C
i2c = I2C(1, scl=Pin(5), sda=Pin(4), freq=400000)
```

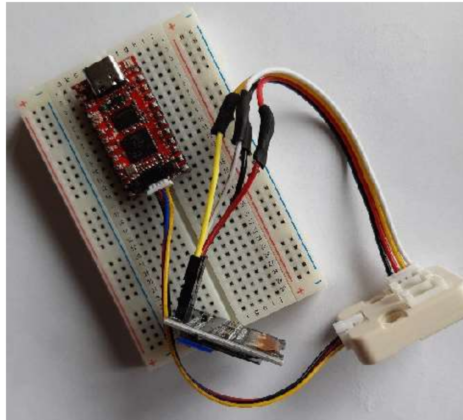
De I2C-poorten van een **ESP8266** werken hardwarematig, je mag elk willekeurig tweetal GPIO's hiervoor gebruiken. Je declareert een I2C poort als

```
from machine import Pin, I2C
i2c = I2C(scl=Pin(5), sda=Pin(4), freq=400000)
```

I2C is een synoniem voor SoftI2C

Grove, QWIIC en Stemma QT connectoren

Veel controller bordjes hebben één van deze connectoren voor aansluiting van I2C-randaparaten. Dat is handiger dan jump-wires. Met één stekker sluit je voeding, GND, sda en scl aan. De pull-up weerstanden zijn geïntegreerd in dat systeem. Adafruit gebruikt Stemma QT, Sparkfun de identieke QWIIC en Seeeduino de grotere Grove stekker. Er bestaan convertorkabels van QWIIC naar Grove. Sommige uitbreidingsbordjes hebben een in- en een uitgang zodat je ze achter elkaar kunt aansluiten en er bestaan splitters voor de I2C signalen. De foto hieronder toont een QWICC/Grove systeem. Een adaptor kabel verbindt de QWICC-poort van de controller met een Grove splitter. Een Grove kabeltje verbindt de splitter met een display.



Figuur 74: Een QWIIC/Grove combinatie

Ga na welke GPIO's verbonden zijn met de I2C-connectoren.

Klasse I2C

Klasse I2C is een onderdeel van library machine en ondersteunt het gebruik van de I2C-bus. De constructor is:

i2c = machine.I2C(*id*, *sda*, *scl*, *freq*)

of, kortweg

i2c = machine.I2C(1,sda=machine.Pin(2), scl=machine.Pin(3), freq=400000)

id is het nummer van de gebruikte I2C-bus, sda en scl zijn GPIO's en freq is de frekwentie van de bus in Herz. Probeer zo veel mogelijk dezelfde combinatie te gebruiken. In dit boek is dat niet gelukt: we gebruiken verschillende bordjes, met andere QWIIC/Grove aansluitingen en andere GPIO configuraties.

In het voorbeeld hieronder zie je hoe we de constructor kunnen gebruiken. **I2C.scan** controleert de bus en heeft als resultaat een list met de I2C-adressen van de verbonden apparaten. In het voorbeeld hieronder definiëren we twee I2C-poorten. Aan de ene hangt een beeldscherm (adres 60) en 2 sensoren, aan de andere alleen een beeldscherm.

```
#------------------------------#
#  i2c-scan.py                 #
#                              #
#  28 september 2021           #
#  Dirk Ghysels                #
#------------------------------#
import machine
from machine import Pin, I2C
i2c0 = I2C(0, scl=Pin(25), sda=Pin(24), freq = 200000)
```

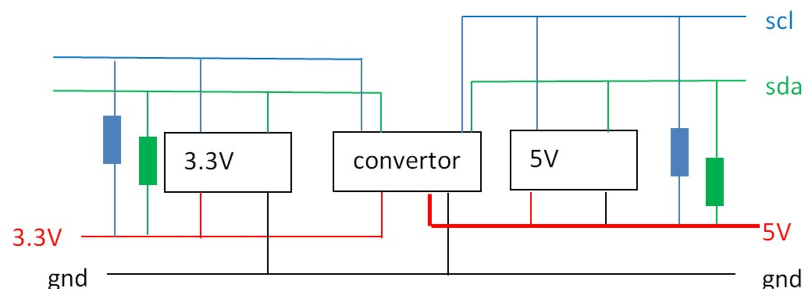
```
i2c1 = I2C(1, scl=Pin(23), sda=Pin(22), freq = 200000)
print(i2c0.scan())
print(i2c1.scan())
```

```
Shell

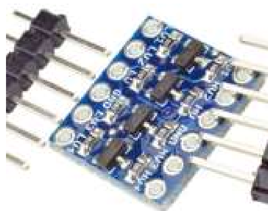
MicroPython v1.17 on 2021-09-02; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
  [60, 64, 81]
  [60]
>>>
```

Figuur 75: i2c-scan

Opgelet: Er zijn I2C onderdelen met signaalniveaus en voedingsspanning van 5 volt, en andere met niveaus van 3.3 volt. Als je die door elkaar gebruikt worden onderdelen beschadigd. Gebruik een **logical level converter** om veilig 5 V en 3.3 V segmenten met elkaar te verbinden. Deze converteert signalen van 0..5V naar signalen van 0..3.3V.



Figuur 76: I2C systeem met 3,3V en 5V systemen



Figuur 77: logical level convertor

Klasse SoftI2C (niet voor alle controllers)

Klasse SoftI2C kan je gebruiken als je de voorgeschreven GPIO’s voor de I2C-bus niet kunt gebruiken of als je meerdan twee identieke I2C-slaves gaat gebruiken. Deze

klasse gebruikt de hardware I2C-bus van de controller niet. De bus wordt softwarematig geïmplementeerd. Het aantal beschikbare GPIO's voor de hardware I2C is zeer groot.